Preprint
UCRL-JC-143204

Combining a Multithreaded Scene Graph System with a Tiled Display Environment

W. Bethel, R. Frank

This article was submitted to
The Institute of Electrical and Electronics Engineers Visualization 2001, San Diego, CA., October 21-26, 2001

March 29, 2001

U.S. Department of Energy
Lawrence
Livermore
National
Laboratory

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Combining a Multithreaded Scene Graph System with a Tiled Display Environment

Wes Bethel
R3vis Corporation

Randall Frank
Lawrence Livermore National Laboratory

1.0 Abstract

This case study highlights the technical challenges of creating an application that uses a multithreaded scene graph toolkit for rendering and uses a software environment for management of tiled display systems. Scene graph toolkits simplify and streamline graphics applications by providing data management and rendering services. Software for tiled display environments typically performs device and event management by opening windows on displays, by gathering and processing input device events, and by orchestrating the execution of application rendering code. These environments serve double-duty as frameworks for creating parallel rendering applications. We explore technical issues related to interfacing scene graph systems with software that manages tiled projection systems in the context of an implementation, and formulate suggestions for the future growth of such systems.

2.0 Background

A ***tiled display surface*** is a single logical display composed of one more physical display devices. We use the term tiled display to refer to surround-style systems, like the CAVE [Cruz-Niera93], and high-resolution projector arrays, such as the Powerwall [UMinn94]. In this discussion, we use the term ***host environment*** to refer to the software that performs tiled display device management. This case study explores use of two such environments: CAVELib[1] and VDL [Schikore00].

CAVELib and VDL share a similar design that make it easy to write reasonbly portable applications that use multiple displays. Both use a configuration file that defines a mapping from logical displays to physical display devices. Both environments perform the mundane tasks of opening suitable drawables on the displays, and initializing OpenGL. When a frame needs to be rendered, the host environment computes the view transformation that corresponds to a given display, and invokes the application "draw function" to perform rendering. The host environment typically invokes all application draw functions in parallel in order to achieve the maximum possible frame rate.

A ***scene graph system*** refers to a set of data structures and associated operations that implement data management and rendering in graphics applications. Scene graph systems vary in features, types of primitives supported and in deployment environment. Some provide event, window and rendering context management as an intrinsic (and sometimes inseparable) part of the system, while others rely on the application for these services.

There have been similar efforts in the past to use scene graph toolkits for rendering management in multiple display environments. [Pape96] describes combining Performer [Rohlf94] and CAVELib. These applications can be best described as Performer applications that use CAVELib to gather input device events. These applications do not use CAVELib to compute the view transformation, nor use CAVELib to invoke application rendering code. All view transformations are computed within the Performer scene graph hierarchy. More recently, [Christianson00] compares use of Java3D and Vega as the underlying scene graph technologies with deployment in a multiple display environment. Frame rates for Java3D were reported to be an order of magnitude slower than the native mode Vega. That study does not use an external host environment, such as VDL or CAVELib, but instead employs the inherent multiple-display capabilities

1. CAVELib is a commercial software package distributed by VRCO, http://www.vrco.com/.

of the scene graph system to perform tiled display management.

Our goal is to use both software systems to their fullest potential. The host environment will manage displays, gather input device events, and compute the view transformation for each display. The scene graph system will serve as a repository for graphics data and will perform rendering when invoked by the host environment. For this experiment, we used OpenRM Scene Graph, an Open Source, native mode, thread-safe, and multistage-parallel scene graph system ([Bethel01], [OpenRM]).

3.0 Implementation Details

In the discussion that follows, we focus on the interface between OpenRM and the host environment. The interfaces consist of resources that are shared between the two technologies. During our development, we encounter obstacles due to assumptions made by both systems concerning resource usage. We present an example that makes extensive use of features from both systems.

3.1 OpenRM and Host Environments

To begin our development, we first identified the "interfaces" bewteen the host environment and OpenRM. We use the term interface in the sense of resources that are shared between the host environment and OpenRM, as opposed to API-style interfaces. These include the OpenGL rendering context, the drawable, the OpenGL transformation stack, and any special post-rendering activities, such as swapping buffers. These interfaces are summarized in the following table.

TABLE 1. Interfaces between Host Environment and OpenRM

Resource	How Used
OpenGL context	Created and "made current" by the host environment.
Drawable	The OpenGL-capable drawable is created by the host environment, and mapped onto the display device.
Transformation Stack	The host environment computes the view transformation that reflects the correct view transformation for a single viewer looking "through" a "window" on each display device.
Post-render actions	Buffer swapping for double-buffered rendering contexts. The host environment may try to coalesce all buffer swaps into a single point in order to have all display devices update simultaneously.

3.2 OpenGL Rendering Contexts

When the application initializes the host environment, the host environment typically initializes each of the individual display devices. For each display device, the host environment locates a suitable OpenGL context then opens a drawable on the display device. According to the rules of OpenGL, an OpenGL rendering context can be active only in one thread at a time. This means that one OpenGL context can be used by multiple threads, but only in round-robin fashion. In order to achieve maximum frame rates, most host environments open one OpenGL rendering context per display, and invoke application draw callbacks in parallel.

When the host environment computes the view transformation for each display prior to invoking the application callback, the host environment writes this transformation into the OpenGL matrix stack. In order to write into the OpenGL matrix stack, the host environment must "own" the OpenGL context. The application rendering callback then draws into the display using the view transformation loaded on the matrix stack. The implication of this design is that the application callback must exist in the same execution thread as the host environment process that computes the view transformation.

3.3 Matrix Stack Management

The RMpipe object in OpenRM has a two-state attribute that controls how the OpenGL matrix

stack is initialized during rendering. In one state, OpenRM will intialize the matrix stack by loading the identity matrix prior to rendering. In the other state, the matrix stack is not initialized: it is assumed the caller has pre-loaded values on the matrix stack, and any transformations contained in the scene graph are concatenated onto the matrix stack. The developer must request the latter mode in order to use OpenRM with a host environment that preloads the view transformation on the matrix stack.

3.4 Who Owns the OpenGL Context?

Sharing an OpenGL context amongst multiple execution threads can be a tricky proposition. Each execution thread must explicitly "make the context current" before use, and must explicitly yield the context when finished. The host environment is a parallel application, and OpenRM is also multithreaded. In order to have both software systems harmoniously coexist, we need rules concerning ownership of the OpenGL context.

OpenRM supports several modes of multistage, multithreaded rendering. In one such mode, one execution thread performs view dependent processing, such as frustum culling and level-of-detail model switching. The other thread performs render operations, dispatching primitives not culled by the view stage to the graphics pipeline. Since the host environments do not yield the OpenGL context prior to invoking the render callback, having the rendering work performed by a separate thread resulted in problems. However, we would like to benefit from multistage processing during rendering, and would like to use the host environment framework for display management.

In order to solve this problem, we added a new processing mode to the RMpipe object. This new mode places the view dependent processing in a separate execution thread, while the render processing remains in the same thread as the host environment. This is possible because OpenRM's view dependent processing does not require access to the OpenGL context: it only needs the initial matrices from the OpenGL matrix stack, which can be provided internally. The multistage rendering model introduces a one-frame latency, so that the view transformation specified by the host environment at frame *N* is not used for rendering until frame *N+1*.

3.5 Example: A Flyover and VDL

For this example, our goal was to combine a terrain flyover demonstration program with VDL, and to display the results on a tiled display surface. The scene graph created by the demonstration program contains a 3D camera: the position and orientation of the camera changes over time. The challenge was to combine VDL, which has its own view model, with this scene graph, which also includes a view model. It turns out that VDL's view transformation consists of two components: a frustum and a shear transformation. The shear transformation aligns the frustum to a particular display window. When we specify an Identify transformation for the frustum component, VDL preloads only the shear transformation onto the matrix stack. When the OpenRM rendering callback is invoked for each display device, OpenRM's view transformation is multiplied with the shear transformation on the matrix stack, producing the correct view transformation. The flyover example uses extensive view-dependent operations to perform frustum culling and distance-based level-of-detail model switching [Clark76].

4.0 Conclusions and Future Work

Host environments simplify resource management in multiple display environments. They open windows on the displays, compute the view transformation for each display device, and invoke application callbacks to perform rendering. Scene graph systems provide graphics data management and rendering services. Combining these two technologies is a logical choice for developers, and doing so requires careful attnetion to detail. Both host environments we have studied store the view transformation on the OpenGL matrix stack. Due to the rules governing ownership of the OpenGL rendering context, use of the matrix stack as a communication mechanism precludes the use of a draw process that runs in a separate execution thread. In the future, host environments should consider approaches that allow for more flexible use of resources.

Our work was performed on SMP systems with multiple graphics pipes. In the future, we will explore deployment of host environments and scene graph toolkits on clusters. Recent advances in parallel graphics APIs [Humphreys00] facilitate

use of distributed memory platforms for tiled display rendering.

5.0 Acknowledgement

The author wishes to thank Lawrence Livermore National Laboratory for providing access to the VDL software and computing facilities. This work was funded by the U.S. Department of Energy through the Small Business Innovation Research (SBIR) program under contract number DE-FG03-00ER83083.

6.0 References

[Bethel01] W. Bethel, Hierarchical Parallelism in a Scene Graph. Submitted to IEEE Symposium on Parallel and Large-Data Visualization and Graphics, March 2001.

[Christianson00] B. Christianson and A. Kimsey, Comparison of Java3D in a Virtual Environment Enclosure (2000) . Master's Thesis, Naval Postgraduate School, Monetery, CA, March 2000.

[Clark76] J. Clark, Hierarchical Geometric Models for Visible Surface Algorithms. Communications of the ACM, 19, 10, pp 547-554. October 1976.

[Cruz-Niera93] C. Cruz-Niera, D. Sandin, T. DeFanti, Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE. In Computer Graphics, *Proceedings of SIGGRAPH 93*, August 1993.

[Humphreys00] G. Humphreys, I. Buck, M. Eldridge and P. Hanrahan, Distributed Rendering for Scalable Displays. In *Proceedings of Supercomputing '00.* Dallas, Texas, November 2000.

[OpenRM] http://openrm.sourceforge.net/

[Pape96] D. Pape, pfCAVE CAVE/Performer Library (CAVELib Version 2.6), http://www.evl.uic.edu/pape/CAVE/prog/pfCAVE.manual.html.

[Rohlf94] J. Rohlf and J. Helman, IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In *Computer Graphics (Proc. ACM Siggraph 94)*, pp 381-394, August 1994.

[Schikore00] D. Schikore, R. Fischer, R. Frank, R. Gaunt, J. Hobson, and B. Whitlock, "High-resolution Multi-projector Display Walls and Applications", IEEE Computer Graphics and Applications, Vol 20, 4:38-44, Jul/Aug, 2000.

[UMinn94] http://www.lcse.umn.edu/research/powerwall/powerwall.html